A Closer Look into the Robustness of Neural Dependency Parsers Using Better Adversarial Examples

Yuxuan Wang[*1] **Wanxiang Che**[1] **Ivan Titov**[2,3] **Shay B. Cohen**[2] **Zhilin Lei**[1] **Ting Liu**[1]
[1]Research Center for Social Computing and Information Retrieval,
Harbin Institute of Technology
[2]ILCC, School of Informatics, University of Edinburgh
[3]ILLC, University of Amsterdam
{yxwang, car, zllei, tliu}@ir.hit.edu.cn
{ititov, scohen}@inf.ed.ac.uk

Abstract

Previous work on adversarial attacks on dependency parsers has mostly focused on attack methods, as opposed to the quality of adversarial examples, which in previous work has been relatively low. To address this gap, we propose a method to generate high-quality adversarial examples with a higher number of candidate generators and stricter filters, and then verify their quality using automatic and human evaluations. We perform analysis with different parsing models and observe that: (i) injecting words not used in the training stage is an effective attack strategy; (ii) adversarial examples generated against a parser strongly depend on the parser model, the token embeddings, and even the specific instantiation of the model (i.e., a random seed). We use these insights to improve the robustness of English parsing models, relying on adversarial training and model ensembling.[1]

1 Introduction

Neural network-based models have achieved great successes in a wide range of NLP tasks. However, recent work has shown that their performance can be easily undermined with adversarial examples that would pose no confusion for humans (Zhang et al., 2020). As an increasing number of successful adversarial attackers have been developed for NLP tasks, the quality of the adversarial examples they generate has been questioned (Morris et al., 2020).

The definition of a valid successful adversarial example differs across target tasks. In semantic tasks such as sentiment analysis (Zhang et al., 2019) and textual entailment (Jin et al., 2020), a valid successful adversarial example needs to be able to alter the prediction of the target model while preserving the semantic content and fluency of the original text. In contrast, in the less explored field of attacking syntactic tasks, the syntactic structure, rather than the semantic content, must be preserved while also maintaining the fluency. Preserving the syntactic structure enables us to use the gold syntactic structure of the original sentence in the evaluation process. While preserving the fluency ensures that ungrammatical adversarial examples, which not only fool the target model but also confuse humans, will not be considered valid. Therefore in this paper, we evaluate the quality of an adversarial example in two aspects, namely the **fluency** and **syntactic structure preservation**.

Recently, Zheng et al. (2020) proposed the first dependency parser attacking algorithm based on word-substitution which depended entirely on BERT (Devlin et al., 2019) to generate candidate substitutes. The rational was that the use of the pre-trained language model will ensure fluency of the adversarial examples. However, we find that using BERT alone is far from enough to preserve fluency.

Therefore, in this paper, we propose a method to generate better adversarial examples for dependency parsing with four types of candidate generators and filters. Specifically, our method consists of three steps: (i) determining the substitution order, (ii) generating and filtering candidate substitutes for each word, (iii) searching for the best possible combination of substitutions, based on pre-computed candidates and the substitution order. We verify the superiority of the proposed method in terms of syntactic structure preservation and fluency using both automatic and human evaluations, and further show the limitation of the previous BERT-based method.

Table 1 shows adversarial examples generated by our and the method of Zheng et al. (2020), demonstrating that examples generated by our method are

[*] Work partially done while at the University of Edinburgh.

[1]Our code is available at: https://github.com/WangYuxuan93/DepAttacker.git

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 2344–2354

Model	Ours	Zheng et al.
example-1	Most of those ~~**freed**~~ **emancipated** had spent at least 25 years in prison.	Most of those freed ~~**had**~~ **were** spent at least 25 years in prison.
example-2	Boeing received a $ 46 million Air Force contract for ~~**developing**~~ **devising** cable systems for the Minuteman Missile.	Boeing received a $ 46 million Air ~~**Force**~~ **America** contract for ~~**developing**~~ **securing** cable systems for the Minuteman Missile.
example-3	He used better than 5,000 words heaping scorn on the ~~**witnesses**~~ **eyewitnesses** for exercising the Fifth.	He used better than 5,000 ~~**words**~~ **times** heaping scorn on the ~~**witnesses**~~ **dollars** for exercising the ~~**Fifth**~~ **grand**.

Table 1: Adversarial examples generated by our and Zheng et al. (2020)'s methods. The original words are highlighted in bold blue font while the substitute words are highlighted in bold red ones.

more fluent, producing complicated substitutes like **emancipated**, **devising** and **eyewitnesses**. Additionally, since it is nontrivial to decode valid multi-subword tokens from BERT, the BERT-based method of Zheng et al. (2020) only generates single subwords as substitutes.

With the proposed attacking method, we evaluate the robustness of different parsing models and analyse the properties of adversarial attacks. We find that (i) the introduction of out-of-vocabulary (OOV, words not in the embedding's vocabulary) and out-of-training (OOT, words not in the training set of the parser) words in adversarial examples are two main factors that harm models' performance; (ii) adversarial examples generated against a parser strongly depend on the type of the parser, the token embeddings and even the random seed.

Adversarial training (Goodfellow et al., 2015), where adversarial examples are added in the training stage, has been commonly used in previous work (Zheng et al., 2020; Han et al., 2020) to improve a parser's robustness. Only a limited number of adversarial examples have been used in such cases, and Zheng et al. (2020) argued that overuse of them may lead to a performance drop on the clean data. However, we show that with improvement in the quality of adversarial examples produced in our method, more adversarial examples can be used in the training stage to further improve the parsing models' robustness without producing any apparent harm in their performance on the clean data. Inspired by our second finding, we propose to improve the parsers' robustness by combining models trained with different random seeds and embeddings. Such methods, which are not targeting specific types of attacks, should improve the capacity to defend against new attacks as compared to standard adversarial training.

2 Method

In this section, we first give a formal definition of a dependency parsing attack. Then we describe the proposed attacking method for dependency parsing, shown in Algorithm 1. It consists of three steps, namely ranking word importance (lines 1-4), generating candidates for substitution (line 7) and searching for the best substitute combination (lines 8-21).

2.1 Problem Definition

Given an input text space $\mathcal{X}$ containing all possible input sentences $\boldsymbol{x}$ and an output space $\mathcal{Y}$ containing all possible dependency trees of $\boldsymbol{x}$, a parser $F : \mathcal{X} \rightarrow \mathcal{Y}$ learns to map the sentence $\boldsymbol{x}$ to its corresponding tree $\boldsymbol{y}$, denoted by $F(\boldsymbol{x}) = \boldsymbol{y}$. The i-th word of $\boldsymbol{x}$ is denoted by x_i. For sentence $\boldsymbol{x}$, a valid adversarial example $\boldsymbol{x}^*$ is crafted by adding a perturbation to $\boldsymbol{x}$ so that

$$F(\boldsymbol{x}^*) \neq \boldsymbol{y}, \sigma(\boldsymbol{x}^*, \boldsymbol{x}) \leq \epsilon,$$

where σ is a constraint function and ϵ ensures that i) the perturbation is imperceptible, ii) the true dependency tree of $\boldsymbol{x}^*$ should be the same as that of $\boldsymbol{x}$. In this paper, these two constraints are ensured through the use of various filters (see Section 2.3) and are used to evaluate the quality of adversarial examples (see details on fluency and syntactic structure preservation in Section 3.3).

2.2 Word Importance Ranking

Word importance ranking in our model is based on the observation that some words have a stronger influence on model prediction than others. Such word importance is typically computed by setting each word to unknown and examining the changes in their predictions (Li et al., 2016; Ren et al., 2019).

Algorithm 1 Dependency Parsing Attack

Input: Sentence example $\boldsymbol{x}^{(0)} = \{x_1, x_2, \ldots, x_N\}$, maximum percentage of words allowed to be modified γ
Output: Adversarial example $\boldsymbol{x}^{(i)}$

for $i = 1$ to N **do**
  Compute word importance $I(\boldsymbol{x}^{(0)}, x_i)$ via Eq. 1
end for
Create a set W of all words $x_i \in \boldsymbol{x}^{(0)}$ sorted by the descending order of their importance $I(\boldsymbol{x}^{(0)}, x_i)$.
$t = 0$
for each word x_j in W **do**
  Build candidate set $\mathcal{C}_j$ for x_j following the Candidate Substitute Generating step
  Initialise valid candidate set $\mathcal{VC} \leftarrow \{\}$
  for each candidate c_k in $\mathcal{C}_j$ **do**
    Compute the accuracy change $S(\boldsymbol{x}^{(t)}, c_k, j)$ via Eq. 3
    if $S(\boldsymbol{x}^{(t)}, c_k, j) \leq 0$ **then** continue **end if**
    Add c_k to the set $\mathcal{VC}$
  end for
  if $\mathcal{VC}$ is not empty **then**
    $c^* = \underset{c \in \mathcal{VC}}{\operatorname{argmax}} S(\boldsymbol{x}^{(t)}, c, j)$
    $t = t + 1$
    $\boldsymbol{x}^{(t)} \leftarrow$ Replace x_j in $\boldsymbol{x}^{(t-1)}$ with c^*
    if $t \geq \gamma \cdot N$ **then return** $\boldsymbol{x}^{(t)}$ **end if**
  end if
end for
if $t > 0$ **then return** $\boldsymbol{x}^{(t)}$ **else return** None **end if**

This helps to determine the word substituting order in the proposed method.

In this work, we use a combination of the changes found in the unlabelled attachment score (UAS) and in the labelled attachment score (LAS) to measure word importance. Specifically, the importance of a word x_i in sentence $\boldsymbol{x}$ is computed as

$$I(\boldsymbol{x}, x_i) = \lambda_{arc}\Delta_{\text{UAS}}(\boldsymbol{x}, \hat{\boldsymbol{x}}_i) + (1 - \lambda_{arc})\Delta_{\text{LAS}}(\boldsymbol{x}, \hat{\boldsymbol{x}}_i), \quad (1)$$

where $\boldsymbol{x} = x_1 x_2 \ldots x_i \ldots x_N$ is the original sentence and $\hat{\boldsymbol{x}}_i = x_1 x_2 \ldots \text{UNK} \ldots x_N$ replaces x_i with an 'unknown' token. Here $\Delta_{\text{UAS}}(\boldsymbol{x}, \hat{\boldsymbol{x}}_i) = \text{UAS}_{F(\boldsymbol{x})} - \text{UAS}_{F(\hat{\boldsymbol{x}}_i)}$ and $\Delta_{\text{LAS}}(\boldsymbol{x}, \hat{\boldsymbol{x}}_i) = \text{LAS}_{F(\boldsymbol{x})} - \text{LAS}_{F(\hat{\boldsymbol{x}}_i)}$ are the changes in UAS and LAS respectively. λ_{arc} is a coefficient that controls the relative importance of dependency arcs and their labels.

2.3 Generation of Substitute Candidates

Generating substitute candidates is a critical step, as it significantly influences the attack success rate and the quality of generated adversarial examples. Zheng et al. (2020) relied entirely on BERT to generate candidates, but this limits the quality of the adversarial examples. To alleviate this problem, we first collect candidate substitutes from four generation methods, then apply filters to discard inappropriate substitutes, ensuring both diversity and quality of the generated candidates.

2.3.1 Generating Process

We collect substitutes from the following methods:

BERT-Based Method: We use BERT to generate candidates for each target word from its context. This method generates only single subwords.

Embedding-Based Method: Following Alzantot et al. (2018), we use word embeddings of Mrkšić et al. (2016)[2] to compute the N nearest neighbours of each target word according to their cosine similarity and use them as candidates.

Sememe-Based Method: The sememes of a word represent its core meaning (Dong and Dong, 2006). Following Zang et al. (2020), we collect the substitutes of the target word x based on the rule that one of the substitutes the senses of x^* must have the same sememe annotations as one of senses of x.

Synonym-Based Method: We use WordNet[3] to extract synonyms of each target word as candidates.

2.3.2 Filtering Process

We apply the following four types of filters to discard candidates which are likely inappropriate, either in terms of syntactic preservation or fluency.

POS Filter: We first filter out substitutes with different part-of-speech (POS) tags from the original word.[4] This filter is essential for preserving the syntactic structure of the sentence.

Word Embedding Similarity Filter: We use the word embeddings of Mrkšić et al. (2016) to compute the cosine similarity between the original word and each of the substitutes in $\mathcal{C}$ and filter out those whose similarities are less than a threshold ϵ_w.[5]

Grammar Checker Filter: We employ an off-the-shelf grammar checker[6] to filter out candidates that may introduce grammar errors. This filter helps to further ensure that the syntactic structure and fluency are preserved.

Perplexity Filter: We employ GPT-2 (Radford et al., 2019) to calculate the perplexity difference

[2]These embeddings are post-processed to ensure that the nearest neighbours are synonyms.
[3]https://wordnet.princeton.edu
[4]We use the off-the-shelf Stanford tagger. (https://nlp.stanford.edu/software/tagger.html)
[5]Non-synonym substitutes often reduce fluency.
[6]https://pypi.org/project/language_tool

between $\boldsymbol{x}$ and $\boldsymbol{x}_i^c$ for each candidate c:

$$\Delta\text{ppl}(\boldsymbol{x}, c, i) = \text{ppl}(\boldsymbol{x}_i^c) - \text{ppl}(\boldsymbol{x}), \quad (2)$$

where $\boldsymbol{x}_i^c$ is $\boldsymbol{x}$ with its i-th word replaced by c, and filter out c whose $\Delta\text{ppl}(\boldsymbol{x}, c, i) > \epsilon_p$.

2.4 Best Substitute Searching

In this step, we greedily search for the best possible combination of substitutions, relying both on the previously created candidate lists and word substitution order. To preserve the syntactic structure of sentences, we forbid replacement of pronouns, articles, conjunctions, numerals, interjections, interrogative determiners and punctuation. Additionally, we set the maximum percentage of words allowed to be modified γ in the experiments to control the modification number.

Specifically, given a sentence $\boldsymbol{x}$, we substitute the words following the order computed in the word importance ranking step. For each target word x_i, we build an adversarial example $\boldsymbol{x}_i^c = x_1 x_2 \ldots c \ldots x_N$ for each of its substitutes c. Then we compute the accuracy change score from $\boldsymbol{x}$ to $\boldsymbol{x}_i^c$ as input to the parser:

$$\begin{aligned} S(\boldsymbol{x}, c, i) =& \lambda_{arc}\Delta_{\text{UAS}}(\boldsymbol{x}, \boldsymbol{x}_i^c) + \\ & (1-\lambda_{arc})\Delta_{\text{LAS}}(\boldsymbol{x}, \boldsymbol{x}_i^c), \end{aligned} \quad (3)$$

where $\Delta_{\text{UAS}}(\boldsymbol{x}, \boldsymbol{x}_i^c) = \text{UAS}_{F(\boldsymbol{x})} - \text{UAS}_{F(\boldsymbol{x}_i^c)}$ and $\Delta_{\text{LAS}}(\boldsymbol{x}, \boldsymbol{x}_i^c) = \text{LAS}_{F(\boldsymbol{x})} - \text{LAS}_{F(\boldsymbol{x}_i^c)}$ are the changes in UAS and LAS, respectively. If the percentage of modified words in the sentence exceeds a threshold γ, we stop the process. Otherwise, we search for a substitute for the next target word.

3 Experimental Setup

3.1 Target Parsers and Token Embeddings

We choose the following two strong and commonly used English parsers, one graph-based, the other transition-based, as target models, both of which achieve performance close to the state-of-the-art.

Deep Biaffine Parser (Dozat and Manning, 2017) is a graph-based parser that scores each candidate arc independently and relies on a decoding algorithm to search for the highest-scoring tree.

Stack-Pointer Parser (Ma et al., 2018) is a transition-based parser that incrementally builds the dependency tree with pre-defined operations.

We used the following four types of token embeddings to study their influence on each parsers' robustness. To focus on the influence of the embeddings, we use only the embeddings as input to the parsers:

GloVe (Pennington et al., 2014) is a frequently used static *word embedding*.

RoBERTa (Liu et al., 2019) is a pre-trained language model based on a masked language modelling object, which learns to predict a randomly masked token based on its context. It produces contextualised *word piece embeddings*.

ELECTRA (Clark et al., 2020) is a pre-trained language model based on a replaced token detection object, which learns to predict whether each token in the corrupted input has been replaced. It produces contextualised *word piece embeddings*.

ELMo (Peters et al., 2018) is a pre-trained language representation model based on *character embeddings* and bidirectional language modelling.

3.2 Datasets and Experimental Settings

We train the target parsers and evaluate the proposed method on the English Penn Treebank (PTB) dataset,[7] converted into Stanford dependencies using version 3.3.0 of the Stanford dependency converter (de Marneffe et al., 2006) (PTB-SD-3.3.0). We follow the standard PTB split, using section 2-21 for training, section 22 as a development set and 23 as a test set.

It is important to note that when converting PTB into Stanford dependencies, Zheng et al. (2020) maintained the copula (linking verbs) as a head when its complement was an adjective or noun.[8] However, since the design objective of Stanford dependency is to maximize dependencies between content words (de Marneffe et al., 2006), a more typical setting is to regard copulas as auxiliary modifiers. Therefore, we first compare with the previous method by performing this step under their settings and further conduct experiments with the typical PTB-SD-3.3.0 dataset for the convenience of follow-up research.

While training the target parsers, we adopt the hyper-parameters from their respective papers. Note that to compare with the biaffine parser, which uses first-order features, we also adopt the basic setting for the stack-pointer parser.[9] When using

[7] https://catalog.ldc.upenn.edu/LDC99T42

[8] Referred to as PTB-SD-3.3.0-COP in the rest of the paper.

[9] According to our preliminary experiments, neither second-order features nor beam search has an obvious influence on the parser robustness under our attack.

RoBERTa, ELECTRA or ELMo embeddings as input, we set the learning rate of these pre-trained models to 2e-5 and that of other parameters to 2e-2.

For the hyper-parameters of each attacking method, we set the word embedding similarity threshold $\epsilon_w = 0.7$, the candidate perplexity difference threshold $\epsilon_p = 20.0$, the arc importance coefficient $\lambda_{arc} = 0.5$ and the maximum percentage of words allowed to be modified $\gamma = 15\%$.

3.3 Evaluation Metrics

As introduced in Section 2.1, two constraints should be satisfied for an adversarial example to be valid: i) the perturbation is imperceptible, ii) the true dependency tree of x^* should be the same as that of x. For the first, we use **fluency** to measure the imperceptibility of the perturbations, and assume that in a fluent adversarial example the perturbation is imperceptible. For the second, **syntactic structure preservation** is used to measure whether an adversarial example's true dependency tree is identical to that of the original text. Both automatic and human evaluations are used for analysis.

In the automatic evaluation, GPT-2 (Radford et al., 2019) is used to compute the average perplexity of the adversarially modified PTB test set to measure the overall fluency. In the human evaluation, we ask three annotators to evaluate the quality of adversarial examples in two aspects, namely syntactic structure preservation and fluency.[10] To evaluate the preservation of the syntactic structure, we randomly collect 100 sentences along with their adversarial examples and ask the annotators to decide whether the syntactic structure is preserved in each case. For the fluency evaluation, we randomly collect 100 sentences along with the adversarial examples generated by our method and those produced by the black-box method of Zheng et al. (2020).[11] For each sentence, the annotators are asked to distinguish which example is better with regard to fluency. For both evaluations, we adopt the majority vote for the final results.

To evaluate how successful the attack is, we report the parsing results of the target models on the original and the adversarially modified (*after-attack*) PTB test set. The results are reported in terms of unlabelled attachment score (UAS) and labelled attachment score (LAS). We also report the attack success rate, namely the percentage of successfully attacked sentences. If the prediction accuracy of the modified sentence is lower than the original one, it is regarded as a successful attack.[12]

4 Results

4.1 Comparison with Previous Work

We first evaluate our attacking method on PTB-SD-3.3.0-COP and compare it with previous work (Zheng et al., 2020). Since we focus on the black-box attack in this paper, we compare with their sentence-level black-box attack against the deep biaffine parser with only word-based embeddings as input. In both their and our settings, 15% of words are allowed to be modified.

Model	Automatic	Human	
	PPL	Fluency%	Syntax%
Zheng et al.	267.96	20	75
Ours	**139.99**	**80**	**85**

Table 2: Automatic and human evaluation results on the PTB-SD-3.3.0-COP test set. PPL denotes the average perplexity. Syntax% denotes the preserved syntactic-structure rate and Fluency% the higher fluency rate.

Table 2 shows that adversarial examples generated by our method substantially outperform the previous method with regard to fluency and syntactic structure preservation. In the automatic evaluation, the average perplexity of examples generated by our method is 139.99, as compared to 267.96 of those generated by the previous work. For comparison, the average perplexity of the original PTB test set is 127.67, which is very close to ours.

In the human evaluation, results show that for 80% of the sentences, our adversarial examples have better fluency, which further confirms the effectiveness of our method. In addition, 85% of the examples we generated preserve the original syntactic structure, as compared to 75% reported by Zheng et al. (2020), showing that our method also improves the syntactic-structure preservation rate.

Table 3 shows the attack results of the two methods.[13] It is clear that with higher quality, the adversarial examples generated by our method cause

[10] The three human annotators are postgraduate students with a few years of research experience in syntactic parsing.

[11] We thank Zheng et al. (2020) for kindly providing us with the adversarial examples they generated.

[12] Note that Zheng et al. (2020) only considered unlabelled scores, so when comparing with these, we use the difference in UAS as the measurement of successful attacks. Conversely, in experiments on PTB-SD-3.3.0, we use the difference in LAS.

[13] We only compare UAS here since they did not report LAS in their paper.

Model	Orig-UAS	After-UAS	Succ%
Zheng et al.	95.52	88.69	51
Ours	95.45	88.95	44

Table 3: Results on the PTB-SD-3.3.0-COP test set. Orig-/After-UAS denotes the original and after-attack UAS respectively. Succ% denotes the success rate.

fewer incorrect predictions. This suggests that some of the previous attacks that were counted as successful may have used invalid adversarial examples which are either ungrammatical or which have actually changed the original syntactic structure.

Generator	BERT	Emb.	Sem.	Syn.
before	28.64	20.54	8.57	1.74
after	0.54	2.06	0.46	0.17
left%	1.89	10.04	5.32	10.03

Table 4: The average number of candidates **before** and **after** filtering generated by BERT-based (**BERT**), embedding-based (**Emb.**), sememe-based (**Sem.**) and synonym-based (**Syn.**) methods respectively. And the percentages of the **left** candidates.

To further demonstrate the limitation of the BERT-based method which the previous work used as the only candidate generator, we count the average number of candidates from our use of different generators before and after filtering. Results in Table 4 show that although the BERT-based method generates the most candidates before filtering, only 1.89% of them are left after the filters are applied. Whereas the left candidate percentage varies from 5% to 10% for the other three generators. The results further verify that the quality of candidates generated by the BERT-based method is worse than that from the embedding-based, sememe-based and synonym-based methods.

Model	After-UAS	Succ%	PPL
pos	72.21	89	326.06
pos+emb	79.87	75	286.92
pos+emb+gra	81.52	71	254.95
pos+emb+gra+ppl	88.95	44	139.99

Table 5: Ablation study of filters. **pos**, **emb**, **gra** and **ppl** stand for POS (part of speech), word embedding similarity, grammar checker and perplexity filters respectively.

To evaluate the ability of the filters, we conduct an ablation study with different combinations of these filters. Results in Table 5 show that the perplexity as well as the attack success rate decreases when more filters are applied. As expected, the greatest perplexity drop is brought by the perplexity filter.

4.2 Robustness Evaluation of Different Models

Input	Original		After-Attack		Succ%
	UAS	LAS	UAS	LAS	
Biaffine					
G.	95.36	93.49	88.69	85.09	55.3
M.	96.29	94.51	90.70	87.67	47.5
E.	97.12	95.38	91.05	87.79	50.6
R.	97.09	95.41	92.14	89.42	46.1
Stack-Pointer					
G.	94.93	93.05	88.26	84.64	52.6
M.	95.69	93.77	89.57	86.49	46.8
E.	96.94	95.19	90.69	87.47	50.3
R.	96.93	95.20	91.58	88.84	45.1

Table 6: Robustness evaluation results. Succ% denotes the success rate (computed based on LAS). **G.**, **M.**, **E.** and **R.** stand for GloVe, ELMo, ELECTRA and RoBERTa respectively.

We evaluate the robustness of the different parsing models introduced in Section 3.1 on PTB-SD-3.3.0 and report the results in Table 6. First of all, when applied to unperturbed sentences, the graph-based deep biaffine parser performs consistently better than the transition-based stack-pointer parser (using the same embeddings). Among the four kinds of embeddings, the word piece-level embeddings (i.e., ELECTRA and RoBERTa) achieve the highest results, while GloVe yields the lowest results.

As for the adversarially modified sentences, we find that the drop in performance is close between the two families of parsers (using the same embeddings), while the attack success rate against the Stack-Pointer parser is slightly higher. In terms of the embeddings, RoBERTa turns out to be the most robust one, which has the lowest attack success rate and achieves the highest performance on the generated adversarial examples. ELMo is also a comparatively robust embedding. We are surprised to find that although ELECTRA achieves similar performance to RoBERTa on clean input data, it performs poorly on the adversarial examples. We hypothesise that this is due to ELECTRA's training objective, i.e. learning to predict whether a token in a corrupted sentence is genuine or not. With this objective, some of our substitutes can be predicted as incorrect tokens, yielding token representations in the space not encountered by the parser in training, and hence damaging its performance. Lastly, GloVe is the most vulnerable embedding. [14]

[14]To evaluate the stability of the attack, for each parsing

Vocab.	Original				After-Attack				Succ%
	UAS	LAS	OOV	OOT	UAS	LAS	OOV	OOT	
50k	95.36	93.49	2	24	87.58	83.73	972	1285	59.5
400k	95.36	93.49	0	15	88.69	85.09	2	906	55.3
400k (T.)	95.36	93.49	0	8	90.06	87.26	0	0	45.5

Table 7: OOV and OOT test results. **Vocab.** stands for vocabulary size, **T.** means filtering out all the candidates that have not appeared in the training set.

4.3 Out-of-Vocabulary and Out-of-Training Words

In this section, we investigate the roles out-of-vocabulary (OOV, words not in the embedding's vocabulary) and out-of-training (OOT, words not in the training set of the parser) words play in dependency parsing attacks. We perform attacks on the Biaffine GloVe models trained with (i) 50k vocabulary (**50k**), (ii) 400k vocabulary (**400k**) and (iii) the same 400k vocabulary but where all candidates not in the training set are filtered out (**400k (T.)**).

The results are shown in Table 7, where we report the attack results along with the number of OOV and OOT words in the adversarially modified words before and after the attack. Firstly, by comparing the OOV and OOT numbers before and after the attack in the **50k** model, we find that words chosen to be replaced are often non-OOV and non-OOT, while their substitutes are often OOV and OOT. Secondly, the comparison between the **50k** and **400k** results shows that when the number of OOV words decreases, the robustness of the model increases. Therefore, it is reasonable to assume that OOV words in adversarial examples cause incorrect predictions. Thirdly, according to the **400k** and **400k (T.)** results, when the number of OOT words in adversarial examples are reduced to 0 by filtering out all the OOT candidates, the attack success rate drops substantially. Therefore, we have reason to believe that unfamiliar OOT words are another factor degrading a parser's performance.

The OOV problem mostly appears in models using word-level embeddings such as GloVe and can be alleviated by simply increasing the vocabulary size. While for the OOT problem, one potential solution is using adversarial training, where a new parser is trained with a mixture of clean training data and adversarial examples.

model in Table 6 we attack another two trained with different random seeds. The experiment shows all the results are stable across seeds.

4.4 Adversarial Training

Previous work (Zheng et al., 2020; Han et al., 2020) used a limited number (from 2,000 sentences to half of the training data) of adversarial examples in adversarial training as (Zheng et al., 2020) argued that overuse of them may lead to a performance drop on the clean data. In this section, we investigate the adversarial training strategies on all the parsing models introduced in Section 3.1. Specifically, we generate adversarial examples for the whole PTB training set and retrain parsers on different amount of adversarial examples along with the original training set. Figure 1 shows that as the number of adversarial examples used in adversarial training increases, the robustness of the models increases accordingly. For most of the models, the increase of robustness stops between 50% and 70% of adversarial examples used.

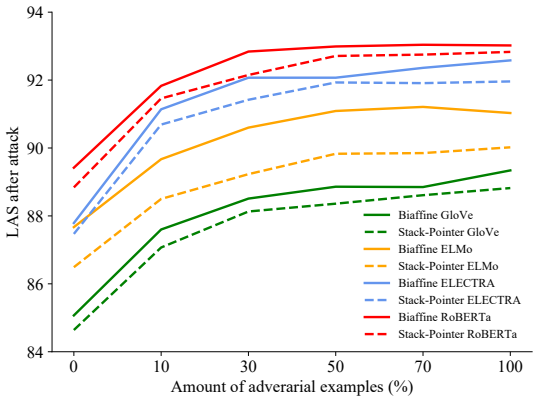

Figure 1: After-attack results with different amounts of adversarial examples used for adversarial training (best viewed in colour).

Table 8 shows the results of Biaffine parsers retrained with 100% of the adversarial examples generated for the original training set. We find that in most cases, the parsing results on the clean data are not obviously influenced although all of the adversarial examples are used. In addition, the robustness of all of the retrained models is substantially improved.

Input	Original		After-Attack		Succ%
	UAS	LAS	UAS	LAS	
Biaffine					
G.	95.36	93.49	88.69	85.09	55.3
G.*	95.32	93.45	91.90	89.34	38.5
M.	96.29	94.51	90.70	87.67	47.5
M.*	96.17	94.37	93.49	91.03	33.2
E.	97.12	95.38	91.05	87.79	50.6
E.*	96.96	95.23	95.03	92.58	33.4
R.	97.09	95.41	92.14	89.42	46.1
R.*	97.03	95.30	95.30	93.02	29.5
Stack-Pointer					
G.	94.93	93.05	88.26	84.64	52.6
G.*	94.92	93.04	91.58	88.82	36.2
M.	95.69	93.77	89.57	86.49	46.8
M.*	95.75	93.81	92.64	90.02	34.1
E.	96.94	95.19	90.69	87.47	50.3
E.*	96.83	95.04	94.53	91.96	34.2
R.	96.93	95.20	91.58	88.84	45.1
R.*	96.80	95.01	95.10	92.83	29.1

Table 8: Adversarial training results. * denotes models with adversarial training.

4.5 Transferability

We refer to adversarial examples as transferable if, generated against one model, they succeed in fooling another one. Previously, Jin et al. (2020) found that in text classification and entailment tasks, adversarial examples are moderately transferable between models with different embeddings. In this section, we examine the following three kinds of transferabilities of adversarial examples in dependency parsing attacks: (i) **Cross Seed**: adversarial examples generated against one model are tested on another model trained with a different random seed; (ii) **Cross Parser**: adversarial examples generated against one model are tested on another from a different family of parsers; and (iii) **Cross Embedding**: adversarial examples generated against one model are tested on another trained with a different type of embedding.

Src	Cross Seed	Cross Parser	Cross Embedding			
			G.	M.	E.	R.
G.	45.0	40.3	**55.3**	27.7	27.2	25.2
M.	36.3	35.6	28.0	**47.5**	29.6	27.2
E.	34.8	40.2	27.1	28.8	**50.6**	29.2
R.	39.8	35.0	25.5	27.9	29.8	**46.1**

Table 9: Attack success rates (%) in the transferability test with Biaffine parser as the source parser. **Src** represents the source model.

Results in Table 9 and 10 show that the attack success rate always drops when adversarial examples are tested on other models, indicating that the adversarial examples strongly depend on the parser model, the token embeddings and even the specific instantiation of the model (i.e., the random seed). Among the three kinds of transferabilities, the cross seed transfer is the strongest while the cross embedding transfer is the weakest.

Src.	Cross Seed	Cross Parser	Cross Embedding			
			G.	M.	E.	R.
G.	41.1	40.8	**52.6**	25.8	25.7	23.7
M.	36.0	34.8	26.5	**46.8**	27.0	26.6
E.	35.3	33.7	24.4	28.5	**50.3**	29.9
R.	41.1	38.8	24.2	26.0	29.7	**45.1**

Table 10: Attack success rates (%) in the transferability test with Stack-Pointer parser as the source parser. **Src.** represents the source model.

4.6 Cross-Seed and Cross-Embedding Ensemble

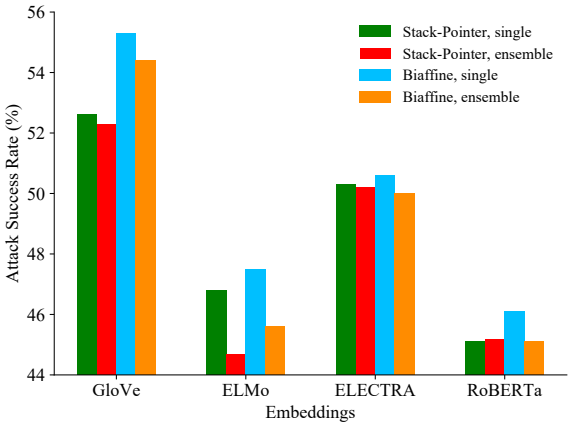

Figure 2: Attack success rates of the examples with and without cross-seed ensemble.

Based on the observations from Section 4.5, we propose to improve the robustness of parsing models using a cross-seed ensemble and cross-embedding ensemble. To ensemble multiple parsers, we simply compute the average of the probability distributions across them and use that result as the new distribution in the ensembled model.

Figure 2 shows the effect of the cross-seed ensemble, where almost all the attack success rates are dropped with such an ensemble. In addition, it is most effective with ELMo while least effective with ELECTRA and RoBERTa.

Table 11 shows the effect of using the cross-embedding ensemble, where robustness increases when more models with different token embeddings are ensembled. Moreover, contrary to adversarial training, the ensemble method is not tuned to specific types of attacks and appears robust to 'unseen' attacks, showing that it is more likely to defend against new attacks.

Input	Original		After-Attack		Succ%
	UAS	LAS	UAS	LAS	
Biaffine					
R.	97.09	95.41	92.14	89.42	46.1
R.G.	97.16	95.55	92.39	89.77	43.8
R.G.M.	97.20	95.58	92.53	89.97	42.5
R.G.M.E.	**97.25**	**95.63**	**92.73**	**90.12**	**41.5**
Stack-Pointer					
R.	96.93	95.20	91.58	88.84	45.1
R.G.	97.01	95.32	92.15	89.49	43.4
R.G.M.	96.98	95.26	**92.28**	89.62	42.3
R.G.M.E.	**97.14**	**95.45**	92.27	**89.64**	**41.5**

Table 11: Cross-embedding ensemble results

5 Related Work

Existing textual adversarial attacks have mostly focused on semantic tasks such as sentiment analysis (Zhang et al., 2019) and textual entailment (Jin et al., 2020). Although most of this work has applied various techniques to maintain the fluency of adversarial examples, a recent study by Morris et al. (2020) reported that quite a number of these techniques introduce grammatical errors.

In syntactic tasks, Zheng et al. (2020) recently proposed the first dependency parser attacking method which depends entirely on BERT to generate candidates. However, we show that the quality of adversarial examples generated by their method is relatively low due to the limitation of the BERT-based generator, and we propose to generate better examples by using more generators and stricter filters.

Han et al. (2020) proposed an approach to attack structured prediction models with a seq2seq model (Wang et al., 2016) and evaluated this model on dependency parsing. They used two reference parsers in addition to the victim parser to supervise the training of the adversarial example generator, and found that the three parsers produce better results when they have different inductive biases embedded to make the attack successful. This finding is quite close in spirit to our conclusion in Section 4.5. Hu et al. (2020) also put forth efforts to modify the text in syntactic tasks while preserving the original syntactic structure. However, their goal is to preserve privacy via the modification of words that could disclose sensitive information.

6 Conclusion

In this paper, we propose a method for generating high-quality adversarial examples for dependency parsing and show its effectiveness based on automatic and human evaluation. We investigate the robustness of different types of neural dependency parsers. We show that OOV and OOT words are two critical characteristics that cause a performance drop and propose to solve the OOT problem with adversarial training. We further examine three kinds of transferabilities of adversarial examples and propose to improve the robustness of parsing models by ensembling across random seeds and token embeddings.

Acknowledgments

This work was supported by the National Key R&D Program of China via grant 2020AAA0106501 and the National Natural Science Foundation of China (NSFC) via grant 61976072 and 61772153.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhendong Dong and Qiang Dong. 2006. *Hownet And the Computation of Meaning*. World Scientific Publishing Co., Inc., USA.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Wenjuan Han, Liwen Zhang, Yong Jiang, and Kewei Tu. 2020. Adversarial attack and defense of structured prediction models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2327–2338, Online. Association for Computational Linguistics.

Zhifeng Hu, Serhii Havrylov, Ivan Titov, and Shay B. Cohen. 2020. Obfuscation for privacy-preserving syntactic parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 62–72, Online. Association for Computational Linguistics.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8018–8025. AAAI Press.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691, San Diego, California. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

John Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. 2020. Reevaluating adversarial examples in natural language. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3829–3839, Online. Association for Computational Linguistics.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148, San Diego, California. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas. Association for Computational Linguistics.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online. Association for Computational Linguistics.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy. Association for Computational Linguistics.

Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. 2020. Adversarial attacks on

deep-learning models in natural language processing: A survey. *ACM Trans. Intell. Syst. Technol.*, 11(3).

Xiaoqing Zheng, Jiehang Zeng, Yi Zhou, Cho-Jui Hsieh, Minhao Cheng, and Xuanjing Huang. 2020. Evaluating and enhancing the robustness of neural network-based dependency parsing models with adversarial examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6600–6610, Online. Association for Computational Linguistics.